# Trading Capacity for Performance in a Disk Array[*]

Xiang Yu[†] Benjamin Gum[†] Yuqun Chen[†]

Randolph Y. Wang[†] Kai Li[†] Arvind Krishnamurthy[‡] Thomas E. Anderson[§]

## Abstract

A variety of performance-enhancing techniques, such as striping, mirroring, and rotational data replication, exist in the disk array literature. Given a fixed budget of disks, one must intelligently choose what combination of these techniques to employ. In this paper, we present a way of designing disk arrays that can flexibly and systematically reduce seek and rotational delay in a balanced manner. We give analytical models that can guide an array designer towards optimal configurations by considering both disk and workload characteristics. We have implemented a prototype disk array that incorporates the configuration models. In the process, we have also developed a robust disk head position prediction mechanism without any hardware support. The resulting prototype demonstrates the effectiveness of the configuration models.

## 1 Introduction

In this paper, we set out to answer a simple question: how do we systematically increase the performance of a disk array by adding more disks?

This question is motivated by two phenomena. The first is the presence of a wide variety of performance-enhancing techniques in the disk array literature. These include striping[17], mirroring[3], and replication of data within a track to improve rotational delay[18]. All of these techniques share the common theme of improving performance by scaling the number of disks. Their performance impacts, however, are different. Given a fixed budget of disks, an array designer faces the choice of what combination of these techniques to use.

The second phenomenon is the increasing cost and performance gap between disk and memory. This increase is fueled by the explosive growth of disk areal density, which is at an annual rate of about 60% [8]. On the other hand, the areal density of memory has been improving at a rate of only 40% per year [8]. The result is a cost gap of roughly two orders of magnitude today.

As disk latency has been improving at about only 10% per year [8], disks are becoming increasingly unbalanced in terms of the relationship between capacity and latency. Although cost per byte and capacity per drive remain the predominant concerns of a large sector of the market, a substantial performance-sensitive (and, in particular, latency-sensitive) market exists. Database vendors today have already recognized the importance of building a balanced secondary storage system. For example, in order to achieve high performance on TPC-C [26], vendors configure systems based on the number of disk heads instead of capacity. To achieve $D$ times the bandwidth, the heads form a $D$-way mirror, a $D$-way stripe, or a RAID-10 configuration [4, 11, 25], which combines mirroring and striping so that each unit of the striped data is also mirrored. What is not well understood is how to configure the heads to get the most out of them.

The key contributions of this paper are:

- a flexible strategy for configuring disk arrays and its performance models,
- a software-only disk head position prediction mechanism that enables a range of position-sensitive scheduling algorithms, and
- evaluation of a range of alternative strategies that trade capacity for performance.

More specifically, we present a disk array configuration, called an SR-Array, that flexibly combines striping with rotational replication to reduce both seek and rotational delay. The power of this configuration lies in that it can be flexibly adjusted in a balanced manner that takes a variety of parameters into consideration. We present a series of analytical models that show how to configure the array by considering both disk and workload characteristics.

[*]This work was supported in part by the Scalable I/O project under the DARPA grant DABT63-94-C-0049 and by the National Science Foundation under grant CDA-9624099 and CCR-9984790.

[†]Department of Computer Science, Princeton University, {xyu,gum,yuqun,rywang,li}@cs.princeton.edu.

[‡]Department of Computer Science, Yale University, arvind@cs.yale.edu.

[§]Department of Computer Science and Engineering, University of Washington, Seattle, tom@cs.washington.edu.

To evaluate the effectiveness of this approach, we have designed and implemented a prototype disk array that incorporates the SR-Array configurations. In the process, we have developed a method for predicting the disk head location. It works on a wide range of off-the-shelf hard drives without special hardware support. This mechanism is not only a crucial ingredient in the success of the SR-Array configurations, it also enables the implementation of rotational position sensitive scheduling algorithms, such as Shortest Access Time First (SATF) [14, 23], across the disk array. Because these algorithms involve inter-disk replicas, without the head-tracking mechanism, it would have been difficult to choose replicas intelligently even if the drives themselves perform sophisticated internal scheduling.

Our experimental results demonstrate that the SR-Array provides an effective way of trading capacity for improved performance. For example, under one file system workload, a properly configured six-disk SR-Array delivers 1.23 to 1.42 times lower latency than that achieved on highly optimized striping and mirroring systems. The same SR-Array achieves 1.3 to 2.6 times better sustainable throughput while maintaining a 15 m$s$ response time on this workload.

The remainder of the paper is organized as follows. Section 2 presents the SR-Array analytical models that guide configuration of disk arrays. Section 3 describes the integrated simulator and prototype disk array that implement the SR-Array configuration models. Section 4 details the experimental results. Section 5 describes some of the related work. Section 6 concludes.

## 2 Techniques and Analytical Models

In this section, we provide a systematic analysis of how a combination of the performance-enhancing techniques such as striping and data replication can contribute to seek distance reduction, rotational delay reduction, overall latency improvement, and throughput improvement. These analytical models, though approximations in some cases, serve as a basis for configuring a disk array for a given workload.

### 2.1 Reducing Seek Distance

We start by defining the following abstract problem: suppose the maximum seek distance on a single disk is $S$, the total amount of data fits on a single disk, and accesses are uniformly distributed across the data set. Then, how can we effectively employ $D$ disks to reduce the average seek latency? We use seek distance to simplify our presentation. (Seek latency is approximately a linear function of seek distance only for long seeks [22].) As a base case, one can show that the average seek distance for reads on a single disk [24] is $S_1 = S/3$.

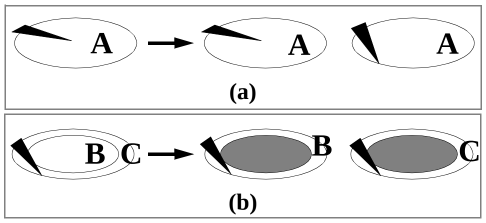


*Figure 1: Techniques for reducing seek distance. Capital letters represent a portion of the data. To the left of the arrows, we show how data is (logically) stored on a single disk. To the right, we show different ways that data on the single disk can be distributed on two disks ($D = 2$): (a) $D$-way mirroring, and (b) $D$-way striping.*

The first seek reduction technique is $D$-way mirroring (shown in Figure 1(a)). $D$-way mirroring can reduce seek distance because we can choose the disk head that is closest to the target sector in terms of seek distance. With $D$ disks, the average seek distance is the average of the minimum of $D$ random variables [3], which is $S/(2D + 1)$.

The second technique is striping (and keeping disks partially empty). Figure 1(b) illustrates a two-way striping. Data on the original single disk is partitioned into two disjoint sets: B and C. We store B on the outer edge of the first disk and C on the outer edge of the second disk. The space in the middle of these two disks is not used. In this case, the single large disk is in effect split into two smaller disks. As a result, the disk head movement is restricted to a smaller region. Assuming constant track capacity and uniform accesses, Matloff [17] gives the average seek distance of a $D$-way stripe ($S_s$):

$$S_s(D) = \frac{S}{3D} \qquad (1)$$

The amount of seek reduction achieved by striping is better than that of $D$-way mirroring. However, $D$-way mirroring provides reliability through the use of multiple copies. A hybrid scheme would provide reliability along with smaller seek latencies. RAID-10, widely used in practice, is a concrete example of such a hybrid scheme: in a RAID-10 system, data is striped across $D_s$ disks while each block is also replicated on $D_m$ different disks.

### 2.2 Reducing Rotational Delay

As we reduce the average seek distance, the rotational delay starts to dominate the disk access cost. To address this limitation, we replicate data at different rotational positions, and by choosing a replica

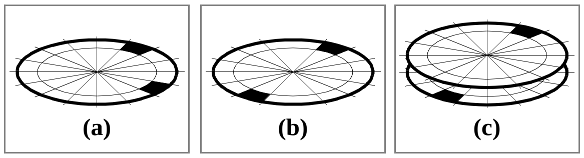


*Figure 2: Techniques for reducing rotational delay. (a) Randomly placed replicas. (b) Evenly spaced replicas. (c) Replicas placed on different tracks (either within a single disk or on different disks).*

that is rotationally closest to the disk head, we can reduce rotational delay. Replication for reducing rotational delay can increase seek distance by pushing data farther apart. We will discuss combining the techniques for reducing seek and rotation distance in a later section.

If the time needed to complete a rotation on a single disk is $R$, we observe that the average rotational delay $R_r(1)$ is simply half of a full rotation, i.e. $R_r(1) = R/2$. If we replicate data $D$ times, and spread the replicas evenly on a track (i.e. $360/D$ degrees apart from each other as shown in Figure 2(b)), the average read rotational latency $R_r$ is:

$$R_r(D) = \frac{R}{2D} \tag{2}$$

We can also show that the average read rotational latency is $R_r = R/(D+1)$, if we randomly place replicas (shown in Figure 2(a)) on the same track. This technique is therefore less beneficial than evenly distributing the replicas and is not used in our design.

However, having multiple replicas on one track increases average rotational latency $R_w$ for writing all these replicas to:

$$R_w(D) = R - \frac{R}{2D} \tag{3}$$

Of course, we could reduce the write costs by writing the closest copy synchronously and propagating other copies during idle periods. Equation (3) gives the worst case cost when we are not able to mask the replica propagation. Notice that $R_r(D) + R_w(D) = R$. Thus if reads are more frequent than writes, making more replicas will reduce overall latency. If reads and writes are equally frequent, varying $D$ will not change the average overall latency. If writes are more frequent than reads, the approach with no replication is always the best. Note that this relationship is independent of the value of R and is only true for foreground replica propagation. Background propagation may make replication desirable even when writes outnumber reads.

Figures 2(a) and (b) illustrate the concept of rotational replication by making copies within the same track. Unfortunately, this decreases the bandwidth of large I/O as a result of shortening the effective track length and increasing track switch frequency. To avoid unnecessary track switches, we place the replicas on different tracks either within a cylinder of a single disk or on different disks (shown in Figure 2(c)). Track skews must be re-arranged so that large sequential I/Os that cross track boundaries do not suffer any unnecessary degradation.

## 2.3 Reducing Both Seek and Rotational Delay

In the previous sections, we have discussed existing techniques for reducing seek distance and rotational delay in isolation. Their combined effects, however, are not well understood. We now develop models that predict the overall latency as we increase the number of disks.

#### SR-Array: Combining Striping and Rotational Replication

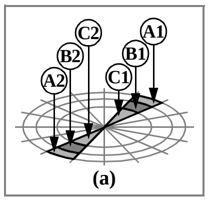
Since disk striping reduces seek distance and rotational replication reduces rotational delay, we can combine the two techniques to reduce overall latency. We call the resulting configuration an *SR-Array.* Figure 3 shows an example SR-Array. In an SR-Array, we perform rotational replication on the same disk. We explore rotational replication on different disks in a later section.

Given a fixed budget of $D$ disks, we would now like to answer the following question: what degree of striping and what degree of rotational replication should we use for the best resulting performance? We call this the “aspect ratio” question. We first consider this question for random access latency, and then we examine how the model can be extended to take into account other workload parameters.

#### Read Latency on an SR-Array

In this paper, we define *overhead* to include various processing times, transfer costs, track switch time, and mechanical acceleration/deceleration times. We focus on the overhead-independent part of the latency in the following analysis.

Let us assume that we have a single disk's worth of data, and we have a total of $D$ disks. Suppose the maximum seek time on one disk is $S$, the time for a full rotation is $R$, only $1/D_s$ of the cylinders on a single disk is used to limit the seek distance, and $D_r$ is the number of replicas for reducing rotational delay ($D_sD_r = D$). If $D_r = 1$, an SR-Array degenerates to simple striping and only $1/D$ of the



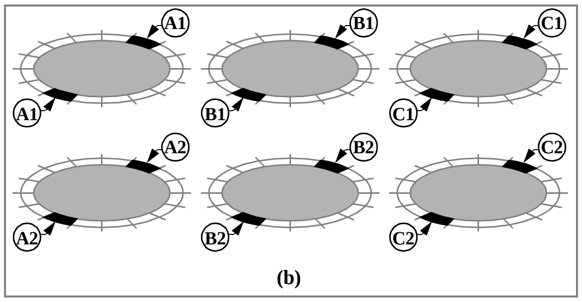


*Figure 3: Reducing both seek and rotational delay in an SR-Array. (a) Data on an original disk is divided into six parts. (b) A 3 × 2 SR-Array. Each disk holds only one sixth of the original data. The two rotational replicas for each block ensure that the maximum rotational delay for any data is half of a full rotation. (Two times the number of disks are needed to support two-way rotational replication; this is shown in the vertical dimension.) The rotational replicas expand the seek distance between different data blocks so the maximum seek distance on each of these six disks is the same as that in a simple three-way striped system (denoted by the three disks in the horizontal dimension).*

available space is used. If $D_s = 1$, we use all the available space. In Figure 3, $D_s = 3$ and $D_r = 2$.

Because the random read latency is the sum of the overhead, the average seek time, and the average rotational time, we can approximate the overhead-independent part of random read latency $T_R(D_s, D_r)$ as:

$$T_R(D_s, D_r) = \frac{S}{3D_s} + \frac{R}{2D_r} \tag{4}$$

Given the constraint of $D_s D_r = D$, we can prove that the following configuration produces the best overall latency for independent random I/Os under low load:

$$\begin{cases} D_s = \sqrt{\frac{2S}{3R}D} \\ D_r = \sqrt{\frac{3R}{2S}D} \end{cases} \tag{5}$$

The overhead-independent part of latency under this configuration is therefore:

$$T_{best} = \sqrt{\frac{2SR}{3D}} \tag{6}$$

It is likely that the optimal $D_s$ and $D_r$ are not integer values. For such scenarios, we choose $D_r$ to be the maximum integer factor of $D$ that is less than or equal to the optimal non-integer value.

Disks with slow rotational speed (large $R$) demand a higher degree of rotational replication. In terms of the SR-Array illustration of Figure 3(b), this argues for a tall thin grid. Conversely, disks with poor seek characteristics (large $S$) demand a large striping factor. In terms of Figure 3(b), this argues for a short fat grid. The model indicates that the latency improvement on an SR-Array is proportional to the square root of the number of disks ($\sqrt{D}$).

So far, our discussion of the model applies to random access by assuming an average seek of $S/3$ in Equation (6). To capture seek locality, we replace $S/3$ with the average seek of a workload. In the later experimental sections, this is accomplished by dividing $S/3$ with a "seek locality index" ($L$), which is observed from the workload. The model does not directly account for sequential access.

## Read/Write Latency on an SR-Array

Now we extend the latency model of an SR-Array to model the performance of both read and write operations. When performing a write, in the worst case scenario of not being able to mask the cost of replica propagation, we must incur a write latency of $T_W(D_s, D_r)$:

$$T_W(D_s, D_r) = \frac{S}{3D_s} + R - \frac{R}{2D_r} \tag{7}$$

Let the number of reads be $X_r$, the number of writes that can be propagated in the background be $X_{wb}$, and the number of writes that are propagated in the foreground be $X_{wf}$. We define the ratio $p$:

$$p = \frac{X_r + X_{wb}}{X_r + X_{wb} + X_{wf}} \tag{8}$$

The average read/write latency, $T(D_s, D_r) = pT_R + (1-p)T_W$, can be expressed as:

$$T(D_s, D_r) = \frac{S}{3D_s} + p\frac{R}{2D_r} + (1-p)(R - \frac{R}{2D_r}) \tag{9}$$

The first term is the average seek incurred by any request. The second term is the average rotational delay consumed by I/O operations that do not result in foreground replica propagation (based on Equation (2)) with probability $p$; and the third term is the rotational delay consumed by writes whose replicas are propagated in the foreground due to lack of idle time (based on Equation (3)) with probability $1-p$. We can prove that the following configuration provides the best overall latency:

$$\begin{cases} D_s = \sqrt{\frac{2S}{3R(2p-1)} D} \\ D_r = \sqrt{\frac{3R(2p-1)}{2S} D} \end{cases} \tag{10}$$

The latency under this configuration is:

$$T_{best} = \sqrt{\frac{2SR(2p-1)}{3D}} + (1-p)R \tag{11}$$

A low $p$ ratio calls for a short fat grid in Figure 3(b). A $p$ ratio under 50% precludes rotational replication and pure striping provides the best configuration. In the best case, when all write replicas can be propagated in the background (or when we have no writes at all), writes and reads become indistinguishable as far as this model is concerned, so $p$ approaches 1 and the latency improvement is proportional to $\sqrt{D}$.

## 2.4 Scheduling and Throughput

We now consider throughput improvements and address the following questions: 1) How do we schedule the requests to take advantage of the additional resources? 2) How do we modify the SR-Array aspect ratio models to optimize for throughput?

### Scheduling on an SR-Array

In our SR-Array design, we choose to place a block and all its replicas (if any) on a single disk. Requests are sent to the only disk responsible for the data, which queues requests and performs scheduling on each disk locally and independently. In contrast, in a mirrored system, because any request can be scheduled for any copy, devising a good global scheduler is non-trivial. We report heuristics-based results for mirrored systems in later sections. In this section, we focus on scheduling for an SR-Array and develop an extension of the LOOK algorithm for an SR-Array, which we call RLOOK.

Under the traditional LOOK algorithm, the disk head moves bi-directionally from one end of the disk to another, servicing requests that can be satisfied by the cylinder under the head. On an SR-Array disk, in addition to scanning the disk like LOOK in the seek direction, our RLOOK scheduling also chooses the replica that is rotationally closest among all the replicas during the scan.

Suppose $q$ is the number of requests to be scheduled for a single RLOOK stroke on a single disk, and $S$, $R$, $D_s$, $D_r$, $D$, and $p$ retain their former definitions from Section 2.3, the average time of a single request in the stroke is $T(D_s, D_r)$:

$$T(D_s, D_r) = \frac{S}{qD_s} + p\frac{R}{2D_r} + (1-p)(R - \frac{R}{2D_r}) \tag{12}$$

The first term amortizes $q$ requests over the end-to-end seek time, which is an approximation of the time needed for a LOOK stroke. The two remaining terms are identical to those of Equation (9). (Empirically, this is a good approximation when $q > 3$. When $q \leq 3$, the requests are so sparse that the latency models of Equations (9) through (11) are used instead.)

Starting with Equation (12), we can prove that the best latency is achieved with the following configuration:

$$\begin{cases} D_s = \sqrt{\frac{2S}{R(2p-1)q} D} \\ D_r = \sqrt{\frac{R(2p-1)q}{2S} D} \end{cases} \tag{13}$$

Under this configuration, the average request latency of RLOOK is:

$$T_{best} = \sqrt{\frac{2SR(2p-1)}{qD}} + (1-p)R \tag{14}$$

Assuming that each request has an overhead of $T_o$, we can approximate the single disk throughput by

$$N_1 = \frac{1}{T_o + T_{best}} \tag{15}$$

In addition to the parameters that we have seen in the previous models, the aspect ratio is now also sensitive to $q$, a measure of the busyness of the system. A long queue allows for the amortization of the end-to-end seek over many requests; consequently, we should devote more disks to reducing rotational delay. In terms of the SR-Array illustration of Figure 3(b), this argues for a tall thin grid. As with the model in the last section, a $p$ ratio under 50% also precludes rotational replication; pure striping is best and Equations (13) through (15) do not apply. In the best case, when all replicas are propagated in the background, $p$ approaches 1, and the model suggests that the overhead-independent part of service time also improves proportionally to $\sqrt{D}$.

Having modeled the throughput of a single disk, we attempt to model the throughput of an SR-Array with $D$ disks and a total of $Q = Dq$ outstanding requests, where $q$ is the average queue size per disk. We assume that the requests are randomly distributed in the system. There could be a load imbalance in the form of idle disks when $Q$ is not much more than $D$. The probability of one disk being idle is $\left(1 - \frac{1}{D}\right)^Q$. Therefore, the total throughput of the system is:

$$N_D \approx D\left[1 - \left(1 - \frac{1}{D}\right)^Q\right] \cdot N_1 \tag{16}$$

Although this approximation is derived based on reasoning about the presence of idle disks, we shall see in Section 4.2 that it is in fact a good approximation of more general cases.

Now that we have described the RLOOK extension to LOOK, it is easy to understand a similar extension to SATF: RSATF. An RSATF scheduler chooses the next request with the shortest access time by considering all rotational replicas. It is well known that SATF outperforms LOOK [14, 23] by considering rotational delay. Our experimental results will show that the gap between RLOOK and RSATF is smaller because both scheduling algorithms consider rotational delays. Once the detailed low level disk layout is understood, RLOOK is simple to implement; it is an attractive local scheduler for an SR-Array.

## 2.5 Comparing SR-Array with Striped Mirror

In an SR-Array, all replicas exist on the same disk. Removing this restriction, we can place these replicas at rotationally even positions on different disks in a "synchronized" mirror, a mirrored system whose spindles are synchronized. We call this layout strategy a *striped mirror*, one flavor of the RAID-10 systems known in the disk array industry. (RAID-10 is a broader term that typically does not necessarily imply the requirement of synchronized spindles and the placement of replicas at rotationally even positions.) To make the performance of the striped mirror competitive to a corresponding SR-Array, we must choose replicas intelligently based on rotational positioning information.

Even with these assumptions, a striped mirror is not equivalent to an SR-Array counterpart. Consider a simple example involving only two disks: blocks A and B reside on different disks in an SR-Array; but each of the disks in a corresponding striped mirror has both blocks. Now consider a reference stream of AAB. On an SR-Array, the two accesses to A are satisfied by two rotational replicas on one disk, consuming less than a full rotation in terms of rotational delay; and the access to B is satisfied by a different disk so its access time is independent of the first disk and the first two accesses. In an attempt to emulate the behavior of the SR-Array, we must send the two accesses to A to the two replicas on different disks in a striped mirror; but now the access time of B is affected by the first two accesses because both disks are busy. In general, it is impossible to enforce identical individual access time for a stream of requests to an SR-Array and a striped mirror.

Statistically, the read latency of a striped mirror should be slightly better than the latency on a corresponding SR-Array. This is because the average of the minimum of the sum of seek and rotational delay is smaller than the sum of the average seek and average minimum rotational delay.

In terms of throughput, the simple example above shows that for an arbitrary stream of requests, there does not exist a general schedule on a striped mirror that is equivalent to that on a corresponding SR-Array. We do not pretend to know how to optimally choose replicas on a striped mirror. Section 3 discusses a number of heuristics. The performance of our best effort implementation of a striped mirror has failed to match that of an SR-Array counterpart.

In terms of feasibility, as spindle synchronization is becoming a rarer feature on modern drives, one can only approximate striped mirrors on unsynchronized spindles. In terms of reliability, a striped mirror is obviously better than an SR-Array.

In general, it is possible to combine an SR-Array with a striped mirror to achieve the benefits of both approaches so that some of the replicas are on the same disk and some are on different ones. The result is the most general configuration: a $D_s \times D_r \times D_m$ "SR-Mirror", where $D_s$ implies that only $1/D_s$ of the space is used (to reduce seek time), $D_r$ is the number of replicas on the same disk, and $D_m$ is the number of replicas on different disks. A $D \times 1 \times 1$ system is $D$-way striping. A $1 \times 1 \times D$ system is a $D$-way mirror. A $D_s \times D_r \times 1$ system is an SR-Array. A $D_s \times 1 \times 2$ is the most common RAID-10 configuration. We may approximate the performance of an SR-Mirror by replacing $D_r$ in the SR-Array models with $D_r \times D_m$.

## 2.6 Summary of Techniques and Models

In this section, we have explored how by scaling the number of disks in a storage system we can 1) reduce seek distance, 2) reduce rotational delay, 3) reduce overall latency by combining these techniques in a balanced manner, and 4) improve throughput. To achieve these goals, the storage system needs to be configured based on a number of parameters. We have developed simple models that capture the following parameters that influence the configuration decisions:

- disk characteristics in the form of seek and rotational characteristics ($S$ and $R$),
- read/write ratio ($p$),
- busyness of the system ($q$), and
- seek locality (L).

We note that there does not exist a single "perfect" SR-Array configuration; instead, there may exist one "right" configuration for every workload and cost/performance specification. As we increase the number of disks, and if we properly configure the storage system, under the right conditions, the various models of this section suggest the following rule of thumb: *By using $D$ disks, we can improve the overhead-independent part of response time by a factor of $\sqrt{D}$.*

## 3 Implementation

In the previous section, we analyzed how to configure a disk array to deliver better performance as we scale the number of disks in the system. In this section, we describe the prototype MimdRAID implementation that puts the theory to test.

### 3.1 Overview

To show the feasibility of our approach, we have developed an infrastructure for experimenting with the various techniques. Figure 4 illustrates the system components and how they stack against each other. There are a number of ways of configuring the system. The controlling agent (at the top) can be either a user level disk driver or an OS kernel device driver. The devices (at the bottom) can be either a disk simulator or real SCSI disks. The remaining components (in the middle) are shared across all configurations and are built in a layered fashion.

The *SCSI Abstraction Layer* abstracts device specific operations such as SCSI device detection at boot time, issuing SCSI commands, and retrieval of command status. We currently support two different 10000 RPM SCSI drives: Seagate ST34502LW and ST39133LWV; but all experimental results reported are based on the ST39133LWV.

The *Calibration Layer* is used for calibrating the disk and extracting information regarding the physical layout of the disk. It keeps track of where the disk head is currently located and calculates how much time is required to move the head from its current position to a target sector. Section 3.2 provides more details about our head-tracking technique.

A parallel layer to the SCSI abstraction layer is the *Simulator*. We decided to integrate the simulator into the architecture to shorten the simulation time for long traces: we not only eliminate idle time, but also replace I/O time (which can be long) with simulated time. The simulator also provides the flexibility of exploring the impact of changing disk characteristics. To faithfully simulate the behavior of the disks that we currently use in the prototype, the simulator receives timing information from the calibration layer to configure itself.

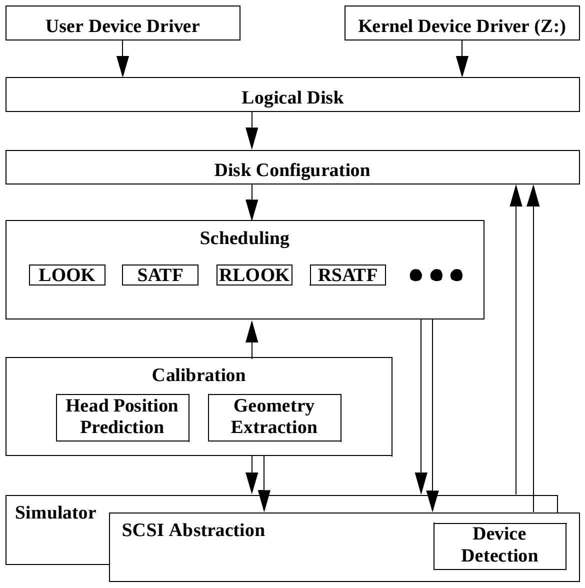


*Figure 4: Prototype architecture.*

The *Scheduling layer* implements several disk scheduling policies. This layer maintains a read/write command queue for each physical disk and invokes a user determined policy to pick the next request at each scheduling step. We call these queues the *drive queues*. Sections 3.3 and 3.4 provide details of the scheduling policies.

The *Disk Configuration Layer* provides support for configuring a collection of disks using techniques such as $D$-way mirroring, striping, SR-Array, and SR-Mirror. It translates I/O requests for a logical disk to a set of I/O commands on the physical disks and inserts them into the appropriate drive queues. The striping unit is 64K bytes in our experiments.

The topmost *Logical Disk Layer* is in charge of exposing the logical disk to the application. The kernel device driver exposes a mount point (e.g. drive letter Z: in Windows 2000) to user space. The user level driver exposes the disk in the form of an API to the application.

### 3.2 Predicting Disk Head Position

The techniques presented in Section 2 rely on the driver's ability to accurately predict the disk head location and the cost of disk operations such as track switches, seeks, and rotational placement. The driver also needs information on the layout of the physical sectors on the disk.

Previous proposals that depend on the knowledge of head positions have relied on hardware support [5, 27]. Unfortunately, this level of support is not always available on commodity drives. We have developed a software-only head-tracking method. Our scheme requires issuing read accesses to a fixed *reference sector* at periodic intervals. The head tracking algorithm computes the disk head position based on the time stamp taken immediately after the completion of the most recent read operation of the reference sector. The basic idea is that the time between two read accesses of the reference sector is always an integral multiple of the full rotation time plus an unpredictable OS and SCSI overhead. By gradually increasing the time interval between adjacent read requests to the reference sector, we amortize the overhead of reading the reference sector. Our experiments show that periodic re-calibration at an interval of two minutes yields predictions that have an error of only 1% of a full rotation time with 98% confidence. It is encouraging that we can achieve a high degree of accuracy with a low overhead associated with reading the reference sector every two minutes. To further reduce this overhead, we can exploit the timing information and known disk head location at the end of a request. We have not implemented this optimization.

To obtain an accurate image of the disk, we use methods that are similar to those used by Worthington [29] for determining the logical to physical sector mapping. We obtain information on disk zones, track skew, bad sectors, and reserved sectors through a sequence of low-level disk operations.

The last piece of information that we measure is the cost of performing track switches and seeks. Small errors in these timing measurements may introduce a penalty that is close to a full rotation. To reduce the number of rotation misses, we introduce a slack of $k$ sectors so that when the mechanism predicts the head to be less than $k$ sectors away from the target, the scheduler conservatively chooses the next rotational replica after the target. This slack can be adjusted by a real time feedback loop to ensure that more than 99% of the requests are on target.

## 3.3 Scheduling Reads

The head-tracking mechanism, along with accurate models of the disk layout and the seek profile, allows us to implement sophisticated local scheduling policies on individual disks; these include RLOOK, SATF, and RSATF.

Scheduling on a mirrored system, however, is more complex due to the fact that a request can be serviced by any one of the disks that have the data. We use the following heuristic scheduling algorithm. When a read request arrives, if some of the disks that contain the data are idle, the scheduler immediately sends the request to the idle disk head that is closest to a copy of the data. If all disks that contain the desired data are busy, the logical disk layer duplicates the request and inserts the copies into the drive queues of all these disks. As soon as such a request is scheduled on one disk, all other duplicate requests are removed from all other drive queues. When a disk completes processing a request, its local scheduler greedily chooses the "nearest" request from its own drive queue. Although this heuristic algorithm may not be optimal, it can avoid load imbalance and works fairly well in practice.

## 3.4 Delayed Writes

While multiple copies of data reduce read latency, they present a challenge for performing writes efficiently because more than one copy needs to be written. We need to make $D_r \times D_m$ copies for a $D_s \times D_r \times D_m$ SR-Mirror. It is however feasible to propagate the copies lazily when the disks are idle. We can issue a write to the closest copy and delay writing the remaining copies. For back-to-back writes to the same data block, which happens frequently for data that die young [21], we can safely discard unfinished updates from previous writes.

In our implementation, we maintain for each disk a *delayed write queue*, which is distinct from the foreground request queue. When a write request arrives, we initially schedule the first write using the foreground request queue just as we do for reads. As soon as writing one of the replicas is scheduled, we set aside the remaining update operations for the other replicas in the individual delayed write queues. Entries from this queue are serviced when the foreground request queue becomes empty. Delayed writes require us to make a copy of the data because the original buffer is returned to the OS as soon as the first write completes.

To provide crash recovery, the physical location of the first write is stored in a *delayed write metadata table* that is kept in NVRAM. Note that it is not necessary to store a copy of the data itself in NVRAM– the physical location of the first write is sufficient for completing the remaining delayed writes upon recovery; so the table is small. When the metadata table fills up to a threshold (10,000 entries in the current implementation), we force delayed writes out by moving them to the foreground request queue.

## 3.5 Validating the Integrated Simulator

So far, we have described the architecture and the components of the integrated MimdRAID simulator and device driver. To establish 1) the accuracy of the head-tracking mechanism, and 2) the validity of

| | |
|---|---|
| Operating system | Microsoft Windows 2000 |
| CPU type | Intel Pentium III 733 MHz |
| Memory | 128 MB |
| SCSI Interface | Adaptec 39160 |
| SCSI bus speed | 160 MB/s |
| Disk model | Seagate ST39133LWV 9.1 GB |
| RPM | 10000 |
| Average seek | 5.2 m$s$ read, 6.0 m$s$ write |

*Table 1: Platform characteristics.*

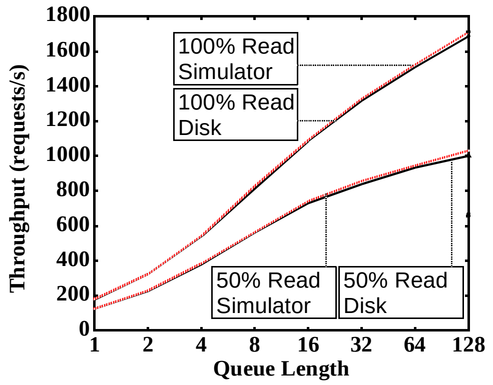


*Figure 5: Comparison of throughput results from the prototype system and the simulator. We use two random workloads, one with just reads, and another with an equal number of reads and writes. The request size is 512 bytes. The array configuration is a $2 \times 3$ SR-Array based on the RSATF scheduler. Writes are synchronously propagated in the foreground. We vary the number of outstanding requests (on the x-axis).*

the simulator, we perform a series of experiments using "Iometer", a benchmark developed by the Intel Server Architecture Lab [13]. Iometer can generate different workloads of various characteristics including read/write ratio, request size, and the maximum number of outstanding requests. We use Iometer to generate equivalent workloads to drive both the device driver and the simulator. Table 1 lists some platform characteristics of the prototype. Figure 5 shows the Iometer result. The throughput discrepancy between the simulator and the prototype under all queueing conditions is under 3%.

To shed more light on the accuracy of the model, in Table 2, we give more detailed statistics of subjecting the model and the prototype to the "Cello base" file system workload (described in Section 4.1). The mean prediction error and low standard deviation show that there are essentially only two types of requests: 99.8% of the predictions are almost right on target, and 0.2% of the predictions miss their targets by a very small amount of time and incur a full rotation penalty. The net effect of these rare rotation misses, however, is insignificant in terms of overall access time. These results indicate that the

| | |
|---|---|
| Misses | 0.22% |
| Mean Prediction Error | 3 $\mu s$ |
| Standard Deviation of Error | 31 $\mu s$ |
| Average Access Time | 2746 $\mu s$ |
| Demerit | 52 $\mu s$ |
| Demerit/Access Time | 1.9 % |

*Table 2: Detailed statistics of model accuracy when subjected to the "Cello base" file system workload. The configuration is a 2×3 SR-Array based on RSATF scheduling. I/O requests in this experiment are physical I/O requests sent to drives; and access time is that of a physical I/O. Prediction error is the difference between the access time predicted by the scheduler and the actual measured access time of a single request. We calculate demerit using the definition by Ruemmler and Wilkes [21].*

simulator faithfully simulates a real SR-Array, allowing us to understand the behavior of the SR-Array using simulation-based results in later sections.

## 4 Experimental Results

In this section, we evaluate the performance of the prototype MimdRAID under two sets of experiments. The first set of experiments is based on playing real-world file system and transaction processing traces on the MimdRAID simulator. The second set of experiments is based on running on the prototype itself a synthetic workload generator that is designed to stress it in ways beyond what is possible with the traces. The purposes of the experiments are to: 1) validate the models of Section 2, 2) show the effectiveness and importance of workload-driven configuration, and 3) demonstrate the use of scaling the number of disks as a cost-effective means of improving performance for certain workloads.

### 4.1 Macro-benchmarks

We test our system using two traces. Cello is a two month trace taken on a server running at HP Labs [21]. It had eight disks and was used for running simulations, compilations, editing, and reading mail and news. We use one week's worth of trace data (for the period of 5/30/92 through 6/6/92). TPC-C is a disk trace (collected on 5/03/94) of an unaudited run of the Client/Server TPC-C benchmark running at approximately 1150 tpmC on a 100 Warehouse database.

#### Logical Data Sets

The 9.1 GB Seagate disks that we use are much larger and faster than any of the original disks used in the trace; therefore, we do not map the original small disks one-to-one onto our large disks. Instead, we group the original data into three logical *data*

|  | Cello base | Cello disk 6 | TPC-C |
|---|---|---|---|
| Data size | 8.4 GB | 1.3 GB | 9.0 GB |
| I/Os | 1,717,483 | 1,545,341 | 3,598,422 |
| Duration | 1 week | 1 week | 2 hours |
| Avg. I/O rate | 2.84/s | 2.56/s | 500/s |
| Reads | 55.2% | 35.8% | 54.8% |
| Async. writes | 18.9% | 16.1% | 0 |
| Seek locality ($L$) | 4.14 | 16.67 | 1.04 |
| Read after write (1 hour) | 4.15% | 3.8% | 14.8% |

*Table 3: Trace characteristics. The "seek locality" row is calculated as the ratio between the average of random seek distances on that disk and the average seek distance observed in the trace. This ratio is used to adjust the $S$ parameter when applying the models of Section 2 in subsequent discussions. The "read after write (1 hour)" row lists the percentage of I/O operations that are reads that occur less than one hour after writing the same data.*

*sets* and study how to place each data set in a disk array made of new disks.

The first data set consists of all the Cello disk data with the exception of disk 6, which houses "/usr/spool/news". We merge these separate Cello disk traces based on time stamps to form a single large trace. The data from different disks are concatenated to form a single data set. We refer to this workload as "Cello base" in the rest of the paper. The second data set consists solely of Cello disk 6. This disk houses the news directory; it exhibits access patterns that are different from the rest of the Cello disks and accounts for 47% of the total accesses. We refer to this workload as "Cello disk 6" in the rest of the paper. The third data set consists of data from 31 original disks of the TPC-C trace. We merge these traces to form a single large trace; and we concatenate these disks to form a single data set as well. We refer to this workload as "TPC-C".

Table 3 lists the key characteristics of the trace data sets. Of particular interest is the last row, which reports the fraction of I/Os that are reads to recently written data. Although this ratio is high for TPC-C, it does not rise higher for intervals longer than an hour. Together with the amount of available idle time, this ratio impacts the effectiveness of the delayed write propagation strategy and influences the array configurations.

To test our system with various load conditions, we also uniformly scale the rate at which the trace is played based on the time stamp information. For example, when the scaling rate is two, the traced inter-arrival times are halved.

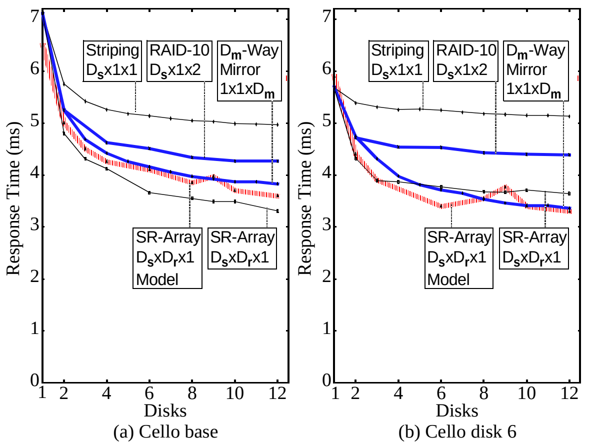


*Figure 6: Comparison of average I/O response time of the Cello file system workloads on different disk array configurations. The SR-Array uses the RSATF scheduler and the remaining configurations use the SATF scheduler. The two configurations labeled as "RAID-10" and "$D_m$-way mirror" are reliable configurations and are denoted by thicker curves. This convention is used throughout the rest of the figures.*

## Playing Cello Traces at Original Speed

As a starting point, we place the Cello base data set on one Seagate disk and the Cello disk 6 data set on another. Although we have fewer number of spindles in this case than in the original trace, the original speed of the Cello traces is still sufficiently low that we are effectively measuring individual I/O latency most of the time. There is also sufficient idle time to mask the delayed write propagations. Therefore, we apply the model in Section 2.3 (Equation (5)) to configure the SR-Array. When applying the formulas, we account for the different degree of seek locality ($L$) in Table 3 by replacing $S$ with $S/L$. We perform replica propagation in the background for all configurations. Although all write operations from the trace are played, we exclude those of asynchronous writes when reporting response time; most of the asynchronous writes are generated by the file system sync daemon at 30 second intervals. All reported response times include an overhead of 2.7 m$s$, which includes various processing times, transfer costs, track switch time, and mechanical acceleration/deceleration times, as defined in Section 2.3.

Figure 6 shows the performance improvement on the Cello workloads as we scale the number of disks under various configurations. The curve labeled as "SR-Array" shows the performance of the best SR-Array configuration for a given number of disks. The SR-Array performs well because it is able to effectively distribute disks to the seek and rotational dimensions in a balanced manner. In contrast, the performance of simple striping is poor due to the

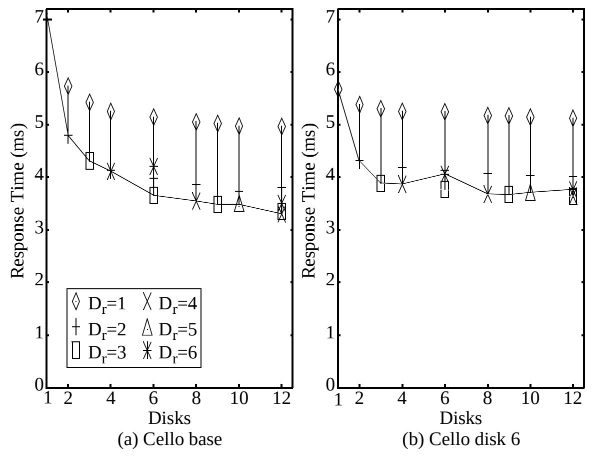


*Figure 7: Configurations of the SR-Array for the two workloads of Figure 6. The curves show the performance of the SR-Array configuration recommended by the model of Equation (5). Each point symbol in the graph shows the performance of an alternative SR-Array configuration with a different number of rotational replicas ($D_r$).*

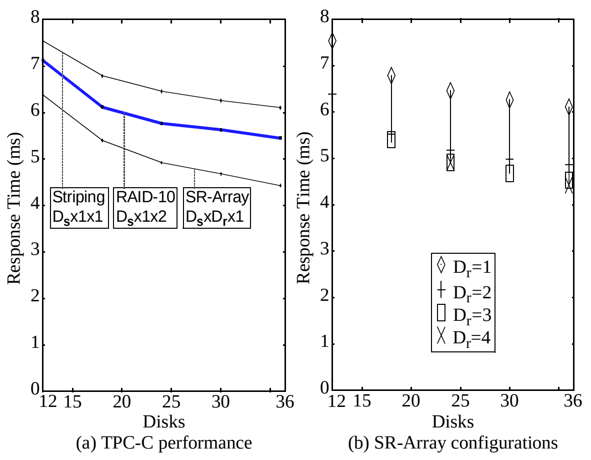


*Figure 8: Average I/O response time of the TPC-C trace. (a) Comparison of striping, RAID-10, and SR-Array. (b) Comparison of alternative configurations of an SR-Array.*

lack of rotational delay reduction. This effect is more apparent for larger numbers of disks due to the diminishing returns from seek distance reduction. The performance of RAID-10 is intermediate because the two replicas allow for a reduction in the rotational delay to a limited extent. $D$-way mirroring is the closest competitor to an SR-Array because of its high degree of flexibility in choosing which replica to read. (We will expose the weakness of $D$-way mirroring in subsequent experiments.) Note that our SATF-based implementation of RAID-10 and $D$-way mirroring are highly optimized versions based on rotational positioning knowledge.

The figure also shows that the latency model of Section 2.3 is a good approximation of the SR-Array performance. The anomalies on the model curves are due to the two following practical constraints: 1) $D_s$ and $D_r$ must be integer factors of the total number of disks $D$, and 2) our implementation restricts the largest degree of rotational replication to six. This restriction is due to the difficulty of propagating more copies within a single rotation, as rotational replicas are placed on different tracks and a track switch costs about 900 $\mu s$. Due to these constraints, for example, the largest practical value of $D_r$ for $D = 9$ is only three, much smaller than the non-integer solution of Equation (5) (5.8 for Cello base and 11.6 for Cello disk 6).

While the Cello base data set consumes an entire Seagate disk, the Cello disk 6 data set only occupies about 15% of the space on a single Seagate disk; so the maximum seek delay of Cello disk 6 is small to begin with for all configurations. Consequently, a larger $D_r$ for an SR-Array is desirable as we increase the number of disks. With these large $D_r$ values, however, the practical constraints enumerated above start to take effect. Coupled with the fact that seek time is no longer a linear function of seek distance at such short seek distances, this explains the slightly more pronounced anomalies of the SR-Array performance with a large number of disks on the Cello disk 6 workload.

Figure 7 compares the performance of other possible SR-Array configurations with that of the configuration chosen by the model. For example, when the number of disks is six, the model recommends a configuration of $D_s \times D_r = 2 \times 3$ for Cello base. The three alternative configurations are $1 \times 6$, $3 \times 2$, and $6 \times 1$. The figure shows that the model is largely successful at finding good SR-Array configurations. For example, on Cello base, with six disks, the SR-Array is 1.23 times as fast as a highly optimized RAID-10, 1.42 times as fast as a striped system, and 1.94 times as fast as the single disk base case.

## Playing the TPC-C Trace at Original Speed

Although a single new Seagate disk can accommodate the entire TPC-C data set in terms of capacity, it cannot support the data rate of the original trace. Indeed, only a fraction of the space on the original traced disks was used to boost the data rate. We start with 12 disks for each of the array configurations. Figure 8 shows the performance as we scale the number of disks beyond the starting point. The data rate experienced by each disk under this workload is much higher than that under the Cello system described in the last section. The workload also contains a large fraction of writes so it also stresses delayed write propagation as idle periods are shorter.

Compared to Figure 6, two curves are missing

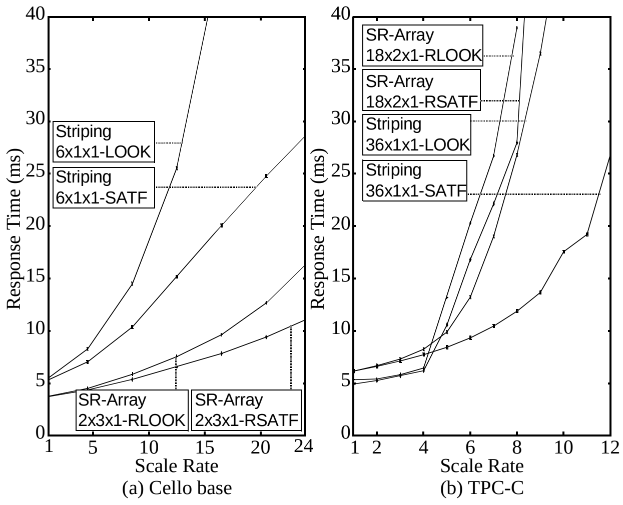


*Figure 9: Comparison of local disk schedulers for different configurations as we raise the I/O rate. We use six disks for Cello base (a), and 36 for TPC-C (b).*

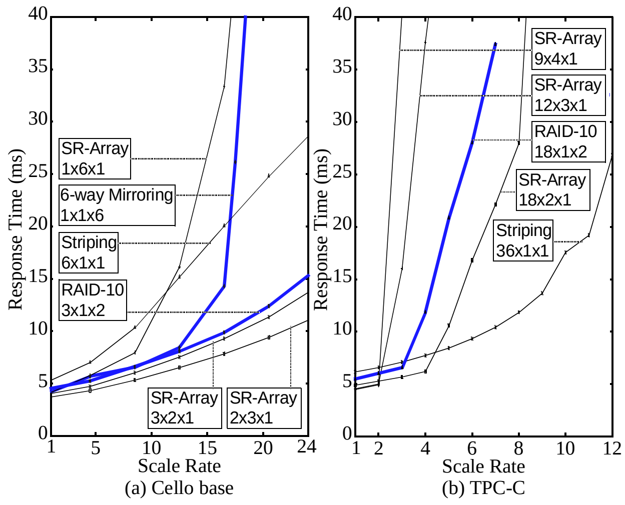


*Figure 10: Comparison of I/O response time on different configurations as we raise the I/O rate. We use six disks for Cello base (a), and 36 for TPC-C (b).*

from Figure 8. One is $D-$way mirroring—it is impossible to support the original data rate while attempting to propagate $D$ replicas for each write. Another missing curve is the latency model—the high data rate renders the latency model inaccurate. The spirit of Figure 8, however, is very much similar to that of Figures 6 and 7: a properly configured SR-Array is faster than a RAID-10, which is faster than a striped system.

What is more interesting is the fact that striping, the only configuration that does not involve replication, is not the best configuration even under the high update rate exhibited by this workload. There are at least two reasons. First, even under this higher I/O rate, there are still idle periods to mask replica propagations. Second, even without idle periods, there exists a tradeoff between the benefits received from reading the closest replicas and the cost incurred when propagating replicas, as demonstrated by the models of Section 2.2; a configuration that can successfully exploit this tradeoff excels. For example, with 36 disks, a $9 \times 4 \times 1$ SR-Array is 1.23 times as fast as a $18 \times 1 \times 2$ RAID-10, and 1.39 times as fast as a $36 \times 1 \times 1$ striped system.

## Playing Traces at Accelerated Rate

Although the original I/O rate of TPC-C is higher than that of the Cello traces, it does not stress the 12-disk arrays discussed in the last section. We now raise the I/O rates to stress the various configurations. For example, when the "scale rate" is two, we halve the inter-arrival time of requests.

Before we compare the different array configurations, we first consider the impact of the local disk schedulers. Figure 9 evaluates four schedulers: LOOK and SATF for striping, and RLOOK and RSATF for an SR-Array. Given a particular request arrival rate, the gap between RLOOK and RSATF is smaller than that between LOOK and SATF. This is because both RLOOK and RSATF take rotational positioning into consideration. Although it is a well known result that SATF out-performs LOOK, we see that SATF alone is not sufficient for addressing rotational delays if the array is mis-configured to begin with. For example, under the Cello base workload, a $2 \times 3 \times 1$ SR-Array significantly out performs a $6 \times 1 \times 1$ striped system even if the former only uses an RLOOK scheduler while the latter uses an SATF scheduler. In the rest of the discussions, unless specified otherwise, we use the RSATF scheduler for SR-Arrays and the SATF scheduler for other configurations.

Figure 10 shows the performance of the various configurations while we fix the number of disks for each workload and vary the rate at which the trace is played. Under the Cello base workload (shown in Figure 10(a)), the 6-way mirror and the $1 \times 6$ SR-Array deliver the lowest sustainable rates. These configurations make the largest number of replicas and it is difficult to mask the replica propagation under high request rates. 6-way mirroring is better than the $1 \times 6$ SR-Array, because the 6-way mirror can afford the flexibility of choosing any disk to service a request, so it can perform better load balancing. The $2 \times 3$ SR-Array is best for all the arrival rates that we have examined; this is because the benefits derived from the extra rotational replicas outweigh the cost. If we demand an average response time no greater than 15 m$s$, the $2 \times 3 \times 1$ SR-Array can support a request rate that is 1.3 times that of a $3 \times 1 \times 2$ RAID-10 and 2.6 times that of a $6 \times 1 \times 1$ striped system.

The situation is different for the TPC-C workload (shown in Figure 10(b)). Under the original trace playing rate, the $9 \times 4 \times 1$ SR-Array is best. As we raise the request arrival rate, we must successively reduce the degree of replication; so the role of the best configuration passes to the $12 \times 3 \times 1$, $18 \times 1 \times 2$, $18 \times 2 \times 1$, and finally, $36 \times 1 \times 1$ configurations, in that order. If we again demand an average response time no greater than 15 m$s$, the $36 \times 1 \times 1$ configuration can support a request rate that is 1.3 times that of a $18 \times 2 \times 1$ configuration and 2.1 times that of a $18 \times 1 \times 2$ RAID-10 configuration.

### Comparison Against Memory Caching

We have seen that it is possible to achieve significant performance improvement by scaling the number of disks. We now compare this approach against one alternative: simply adding a bigger volatile memory cache. The memory cache performs LRU replacement. Synchronous writes are forced to disks in both alternatives. In the following discussion, we assume that the price per MB ratio between memory and disk is $M$. At the time of this writing, 256 MB of memory costs \$300, an 18 GB SCSI disk costs \$400, and these prices give an $M$ value of 57.

Figure 11(a) examines the impact of memory caching on the Cello base workload. At the trace scale rate of one, we need to cache an additional 1.5%, or 126 MB, of the file system in memory to achieve the same performance improvement of doubling the number of disks; and we need to cache 4%, or 336 MB, of the file system to reach the performance of a four-disk SR-Array. $M$ needs to be less than 67 and 75 respectively in order for memory caching to be cost effective, which it is today.

At the trace scale rate of three, using similar reasoning, we can conclude that $M$ needs to be less than 20 in order for memory caching to be more cost effective than doubling the number of disks. Beyond this budget, at this I/O rate, the diminishing locality and the need to flush writes to disks make the addition of memory less attractive. The addition of disks, however, speeds up all I/O operations, albeit at a diminishing rate.

Figure 11(b) examines the impact of memory caching on the TPC-C workload, which has much less locality. We start with a 12-disk SR-Array. At a scale rate of one, $M$ needs to be less than 80 in order for memory caching to be a cost effective alternative to increasing the number of disks to 18 or 24. Adding memory is a more attractive alternative.

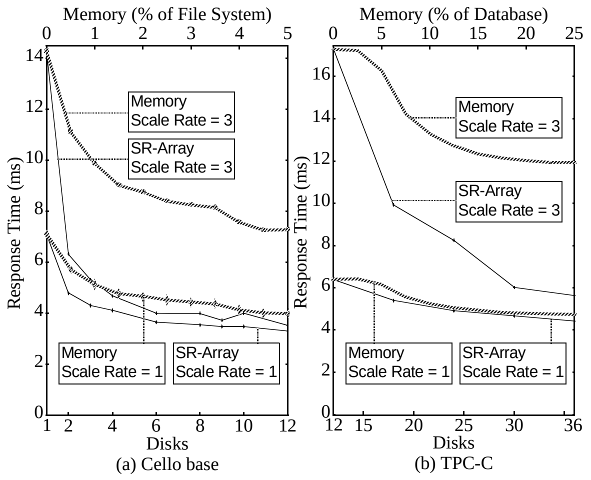


*Figure 11: Comparison of the effects of memory caching and scaling the number of disks. The two SR-Array curves show the performance improvement achieved by scaling the number of disks (bottom x-axis) and they correspond to playing the traces at the original speed and three times the original speed. The two Memory curves show the performance improvement achieved by scaling the amount of memory caching (top x-axis).*

At a scale rate of three, $M$ needs to be less than 24 for memory caching to be more cost effective than increasing the number of disks to 18. Beyond this budget, adding memory provides little additional performance gain, while increasing the number of disks from 18 to 36 can provide an additional 1.76 times speedup.

## 4.2 Micro-benchmarks

To further explore the behavior of the prototype, we use the Intel Iometer benchmark to stress some array configurations in a controlled manner. In all the following micro-benchmarks, we use a seek locality index of 3, as defined in Section 2.3. We measure the throughput in these experiments.

### Throughput Models

In this experiment, we perform only random read operations on the disk array while maintaining a constant number of outstanding requests. (We examine writes more fully in the next subsection.) The goals are 1) to understand the scalability of the disk array, 2) to understand the behavior of the system under different load conditions, and 3) to validate (part of) the throughput model of Section 2.4.

Figure 12 shows that the SR-Array using the RSATF scheduler scales well as we increase the number of disks under this Iometer workload. The RLOOK scheduler is a close approximation of the RSATF scheduler; and the RLOOK-based throughput model closely captures the behavior of the SR-

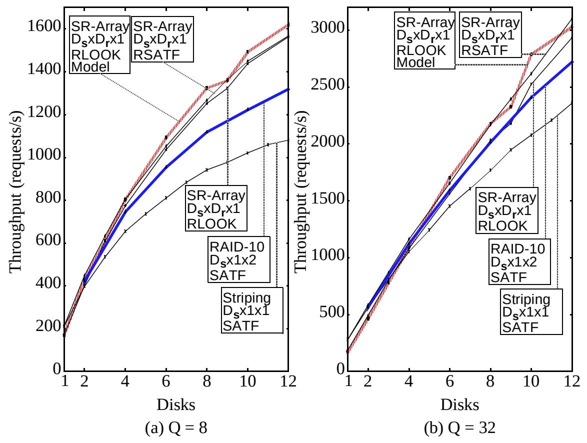


*Figure 12: Throughput as a function of array configuration, number of disks, and queue length. The queue lengths are (a) 8, and (b) 32.*

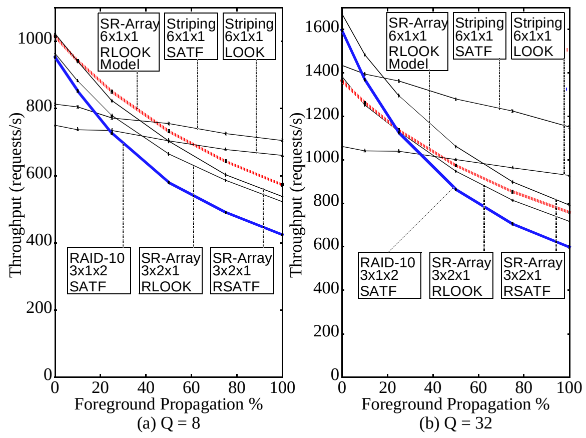


*Figure 13: Throughput as a function of foreground write propagation rate and queue length. The total number of disks is six. The queue lengths are (a) 8, and (b) 32.*

Array, including the throughput degradation experienced when the queue length is short.

The SATF-based striped and RAID-10 systems do not scale as well as the SR-Array. The throughput gap between all these systems, however, narrows as the queue length increases since the SATF scheduler can overcome the lack of rotational replicas when it has a large number of requests to choose from.

**Replica Propagation Cost**

We now analyze configurations under *foreground* write propagation and validate the model in Section 2.4.

Figure 13 shows the throughput results as Iometer maintains a constant queue length of mixed reads and writes. Each write leads to immediate replica propagations; so write ratio and foreground write ratio are the same, namely, $1 - p$, where $p$ is defined by Equation (8) of Section 2.3.

Among the configurations shown in the figure, RAID-10 has the worst performance under high write ratios. To understand why, consider the propagation of a single write: the $3 \times 2 \times 1$ SR-Array requires a single seek followed by writing 2 rotational replicas in a single cylinder; but a corresponding $3 \times 1 \times 2$ RAID-10 requires 2 seeks so the amount of arm movement tends to be greater.

The performance of the striped $6 \times 1 \times 1$ configurations degrade slightly for high write ratios as writes are slightly more expensive than reads.

The performance difference between a $3 \times 2 \times 1$ SR-Array and a $6 \times 1 \times 1$ striped system depends on the write ratio with the former better for low write ratios. If we only consider rotational delay, the rotational replication model of Section 2.2 would imply that the cross-over point between them under LOOK/RLOOK scheduling should be close to the 50% write ratio. If we also consider seek distance, the $3 \times 2 \times 1$ SR-Array has worse seek performance so the actual cross-over point is less than 50%.

Because SATF benefits a $6 \times 1 \times 1$ configuration more than RSATF does to a $3 \times 2 \times 1$ configuration, the cross-over point between these systems under SATF/RSATF scheduling is to the left of that under LOOK/RLOOK scheduling. This distance is even greater when the queue is longer (in Figure 13(b)).

The figure also shows that the RLOOK throughput model (Equation (16)) closely tracks the experimental result under varying write ratios.

## 5 Related Work

A number of previous storage systems were designed to take into consideration the tradeoff between capacity and performance. Hou and Patt [10] performed a simulation study of the tradeoff between mirroring and RAID-5.

The HP AutoRAID incorporated both mirroring and RAID-5 into a two-level hierarchy [28]. The mirrored upper level provided faster small writes at the expense of consuming more storage, while the RAID-5 lower level was more frugal in its use of disk space. Its primary focus was solving the small write problem of RAID-5.

We have taken the tradeoff between capacity and performance a step further by 1) improving latency and throughput of all I/O operations, 2) being able to benefit from more than twice the excess capacity, and 3) providing a means of systematically configuring the extra disk heads.

The HP Ivy project [15] was a simulation study of how a high degree of replication could improve read performance. Our study differs from Ivy in several ways. First, Ivy only explored reducing seek distance and left rotational delay unresolved. Second, Ivy only examined mirroring. The third difference is a feature of Ivy that we intend to incorporate into our system in the future: Ivy dynamically chose the candidate and the degree of replication by observing access patterns. We are currently researching a wide range of access patterns (including those at the file system level) that can be used to dynamically tune the array configuration.

Matloff [17] derived a model of linear improvement of seek distance as one increased the number of disks devoted to striping. Bitton and Gray derived a model of seek distance reduction [3] and studied seek scheduling [2] for a $D$-way mirror. Neither study considered the impact of rotational delay.

Dishon and Liu [6] considered latency reduction on either synchronized or unsynchronized D-way mirrors. A synchronized mirror can reduce foreground propagation latency because the multiple copies can be written at nearly the same time if we insist that the replicas are placed at rotationally identical positions. This advantage comes at the cost of poor read latency because it allows no rotational delay reduction for reads.

Polyzois [20] proposed careful scheduling of delayed writes to different disks in a mirror to maximize throughput, a technique that can potentially benefit delayed writes in our systems when the replicas are on different disks.

The "distorted mirror" [19] provided an alternative way of improving the performance of writes in a mirror. It performed writes initially to rotationally optimal but variable locations and propagated them to fixed locations later. This technique can be integrated with our delayed write strategy as well.

Lumb et al. [16] exploited "free bandwidth" that is available when the disk head is in between servicing normal requests in a busy system. The free bandwidth was used for background I/O activity. Propagating replicas in our system is a good use of this free bandwidth.

Ng examined intra-track replication as a means of reducing rotational delay [18]. We extend this approach to improve large I/O bandwidth by performing rotational replication across different tracks.

The importance of reducing rotational delay has long been recognized. Seltzer and Jacobson independently examined a number of disk scheduling algorithms that take rotational position into consideration [14, 23]. Our work considers the impact of reducing rotational delay in array configurations in a manner that balances the conflicting goal of reducing seek and rotational delay at the same time.

At the time of this writing, the Trail system [12] independently developed a disk head tracking mechanism that is similar to ours. Trail used this information to perform fast log writes to carefully chosen rotational positions. A similar write strategy was in use in the earlier Mime system [5], but Mime relied on hardware support for its rotational positioning information. Aboutabl et al. developed a similar disk timing measurement strategy, which was used to model the response time of individual I/O requests [1].

A number of drive manufacturers have incorporated SATF-like scheduling algorithms in their firmware. An early example was the HP C2490A [9]. Our host-based software solution enables the employment of such scheduling on drives that do not support it internally. Furthermore, it allows experimentation with strategies such as rotational replica selection, strategies that would have been difficult to realize even on drives that support intelligent scheduling internally. On the other hand, if the drive does support intelligent internal scheduling, an interesting question that this study has not addressed is how we can adapt our algorithm for such drives without relying on complex predictions.

One of our goals of studying the impact of altering array configurations is to understand how to configure a storage system given certain cost/performance specifications. The "attribute-managed storage" project [7] at HP shares this goal, although its focus is at the disk array level as opposed to individual drive level.

## 6 Conclusion

In this paper, we have described a way of designing disk arrays that can flexibly reduce seek and rotational delay in a balanced manner. We have presented a series of analytical models that take into consideration disk and workload characteristics. By incorporating these models and a robust software-based disk head position prediction mechanism, the MimdRAID prototype can deliver latency and throughput results unmatched by conventional approaches.

## Acknowledgement

We would like to thank Doug Clark, Ed Grochowski, Ed Lee, Spencer Ng, Chandu Thekkath, Honesty Young, and the class participants of Princeton CS598e (spring 2000) for the early discussions on this topic, HP Labs for supplying the I/O traces, the OSDI reviewers for their comments, and John Wilkes for a large number of excellent suggestions during a meticulous and tireless shepherding process.

## References

[1] Aboutabl, M., Agrawala, A., and Decotignie, J.-D. Temporally Determinate Disk Access: An Experimental Approach (Extended Abstract). In *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Madison, Wisconsin, June 1998), pp. 280–281.

[2] Bitton, D. Arm Scheduling in Shadowed Disks. In *Proc. of 34th IEEE COMPCON* (San Francisco, CA, February 1989), pp. 132–136.

[3] Bitton, D., and Gray, J. Disk Shadowing. In *Proc. of the Fourteenth International Conference on Very Large Data Bases* (Los Angeles, CA, August 1988), Morgan Kaufmann, pp. 331–338.

[4] Borr, A. Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing. In *Proc. of the Seventh International Conference on Very Large Data Bases* (Cannes, France, September 1981), IEEE Press, pp. 155–165.

[5] Chao, C., English, R., Jacobson, D., Stepanov, A., and Wilkes, J. Mime: a High Performance Parallel Storage Device with Strong Recovery Guarantees. Tech. Rep. HPL-CSP-92-9 rev 1, Hewlett-Packard Company, Palo Alto, CA, March 1992.

[6] Dishon, Y., and Lui, T. S. Disk Dual Copy Methods and Their Performance. In *Proc. of Eighteenth International Symposium on Fault-Tolerant Computing (FTCS-18)* (Tokyo, Japan, 1988), IEEE CS Press, pp. 314–318.

[7] Golding, R., Shriver, E., Sullivan, T., and Wilkes, J. Attribute-managed Storage. In *Workshop on Modeling and Specification of I/O* (San Antonio, TX, October 1995).

[8] Growchowski, E. Emerging Trends in Data Storage on Magnetic Hard Disk Drives. In *Datatech* (September 1988), ICG Publishing, pp. 11–16.

[9] Hewlett-Packard Company, Palo Alto, CA. *HP C2490A 3.5-inch SCSI-2 Disk Drives Technical Reference Manual (HP Part No. 5961-4359), 3rd edition.* Boise, Idaho, 1993.

[10] Hou, R., and Patt, Y. N. Trading Disk Capacity for Performance. In *Proc. of the Second International Symposium on High Performance Distributed Computing* (Spokane, WA, July 1993), pp. 263–270.

[11] Hsiao, H.-I., and DeWitt, D. J. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proc. of the 1990 IEEE International Conference on Data Engineering* (February 1990), pp. 456–465.

[12] Huang, L., and Chiueh, T. Trail: Write Optimized Disk Storage System. http://www.ecsl.cs.sunysb.edu-/trail.html.

[13] Intel Server Architecture Lab. Iometer: The I/O Performance Analysis Tool for Servers. http://-developer.intel.com/design/servers/devtools/iometer.

[14] Jacobson, D. M., and Wilkes, J. Disk Scheduling Algorithms Based on Rotational Position. Tech. Rep. HPL-CSP-91-7rev1, Hewlett-Packard Company, Palo Alto, CA, February 1991.

[15] Lo, S.-L. Ivy: A Study on Replicating Data for Performance Improvement. Tech. Rep. HPL-CSP-90-48, Hewlett-Packard Company, Palo Alto, CA, December 1990.

[16] Lumb, C., Schindler, J., Ganger, G. R., Riedel, E., and Nagle, D. F. Towards Higher Disk Head Utilization: Extracting "Free" Bandwidth from Busy Disk Drives. In *Proc. of the Fourth Symposium on Operating Systems Design and Implementation* (San Diego, CA, October 2000).

[17] Matloff, N. S. A multiple disk system for both fault tolerance and improved performance. *IEEE Transactions on Reliability R-36*, 2 (June 1987), 199–201.

[18] Ng, S. W. Improving disk performance via latency reduction. *IEEE Transactions on Computers 40*, 1 (January 1991), 22–30.

[19] Orji, C. U., and Solworth, J. A. Doubly Distorted Mirrors. In *Proc. of ACM SIGMOD Conference* (May 1993), pp. 307–316.

[20] Polyzois, C., Bhide, A., and Dias, D. Disk Mirroring with Alternating Deferred Updates. In *Proc. of the Nineteenth International Conference on Very Large Data Bases* (Dublin, Ireland, 1993), Morgan Kaufmann, pp. 604–617.

[21] Ruemmler, C., and Wilkes, J. UNIX Disk Access Patterns. In *Proc. of the Winter 1993 USENIX* (San Diego, CA, Jan. 1993), Usenix Association, pp. 405–420.

[22] Ruemmler, C., and Wilkes, J. An Introduction to Disk Drive Modeling. *IEEE Computer 27*, 3 (March 1994), 17–28.

[23] Seltzer, M., Chen, P., and Ousterhout, J. Disk Scheduling Revisited. In *Proc. of the 1990 Winter USENIX* (Washington, D.C., Jan. 1990), Usenix Association, pp. 313–323.

[24] Teorey, T. J., and Pinkerton, T. B. A comparative analysis of disk scheduling policies. *Communications of ACM 15*, 3 (March 1972), 177–184.

[25] Teradata Corp. *DBC/1012 Database Computer System Manual Release 2.0*, November 1985.

[26] Transaction Processing Performance Council. *TPC Benchmark C Standard Specification.* Waterside Associates, Fremont, CA, August 1996.

[27] Wang, R. Y., Anderson, T. E., and Patterson, D. A. Virtual Log Based File Systems for a Programmable Disk. In *Proc. of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, LA, February 1999), Operating Systems Review, Special Issue, pp. 29–43.

[28] Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. The HP AutoRAID Hierarchical Storage System. *ACM Transactions on Computer Systems 14*, 1 (February 1996).

[29] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. On-Line Extraction of SCSI Disk Drive Parameters. In *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Ottawa, Canada, May 1995), Performance Evaluation Review 23(1), pp. 146–156.